# Spreadsheets — heaven or hell?

Spreadsheets have found widespread use in most scientific applications, as they are very easy to use for manipulating data. Unfortunately, it is just as easy to make mistakes. Bob McDowall, Chris Burgess and Bill Hardcastle discuss…

R.D. McDowall
Column Editor

You know the scenario, you're walking through the laboratory and pass a colleague working on a spreadsheet on a PC. Looking over their shoulder you notice that they're working on a problem similar to the one that you also have. Stopping to talk you discover that the spreadsheet they're using is just what you want*/can be easily modified* (*delete whichever is not applicable) and your colleague e-mails or gives you a copy of their spreadsheet on disk. What starts as a spreadsheet for an individual, can soon become a departmental standard that everyone is using. Of course, this doesn't happen in your laboratory, does it?

## Ms_Calculation@Itsnotmyfault.guv

Let's roll back the clock to the time when the spreadsheet was originally written. The original purpose may have been for a single individual to perform some informal calculations. However, let's ask some simple questions:

- Is there a specification or similar design document?
- Have the calculations been defined and tested?
- Are the cells locked to prevent changes to the calculations?

If the answer to any of these questions is 'no', you have a potential spreadsheet problem on your hands.

Spreadsheet use must be controlled in any scientific discipline. Although seen as 'restricting scientific freedom and creativity' (a poor argument at the best of times), spreadsheets should be controlled, otherwise the mistakes that can occur from unrestricted, naive and inappropriate use are far greater and can impact both personal, laboratory and organizational credibility far more than if the spreadsheet is not controlled. Don't believe us? Please read on…

The purpose of this column is to highlight some of the problems that can occur with the use of spreadsheet applications and to offer advice on how these might be eliminated, or at least minimized.

## Definitions

For the purpose of the discussion we need to define the following terms.

- Spreadsheet: this is the program, such as Excel, 123 or Quattro Pro. The program is simply an empty shell containing the basic calculation functions needed to develop a spreadsheet application.
- Spreadsheet application: consisting

of a particular instance of a spreadsheet containing data, together with operators or formulae that act on that data. This can include 'macros' (written in Visual Basic or by capturing keystrokes) or templates (your calculations are embedded within a blank spreadsheet, you retrieve the file, rename it and enter your data).

- Application developers: persons who write or encode spreadsheets.
- Application users: persons who use the spreadsheets.

In practice, the developer and user may well be one and the same person; this is especially so given the ease with which spreadsheets can be developed and used.

## What is a spreadsheet?

An electronic spreadsheet consists of an array of cells arranged on a rectangular grid. The cells may contain numeric data, text strings or formulae, and each cell has a unique reference determined by its location on the grid. Along with its cellular structure, a spreadsheet also has a large collection of operators and functions for manipulating the contents of the cells.

In addition, most modern spreadsheets permit the user to combine these operators and functions into new routines — known as 'macros' — for performing specific routine tasks. Once assembled, a couple of keystrokes or a single click of a mouse button can invoke a macro at any time thereafter. This combination of properties makes the spreadsheet a very powerful tool for processing data.

Spreadsheets can also have sort and date calculation functions. This will be discussed later with the Year 2000 issues.

## Power to the user

One of the great attractions of the spreadsheet in the eyes of many is that it removes the need for people to become programmers in order to get a computer to process their data.

Who needs a professional programmer? No point waiting around and writing those User Requirements Specifications (URS) — fingers on the keyboard and away you go.

Once written, entering data is easy — they can be typed straight into any of the cells. Doing something useful with data once they are entered is also easy, thanks to the built-in collection of operators and functions. The functions act like mini-programs and allow the user to 'pick-and-mix' in order to obtain the desired results. These can range from:

- routine calculations
- exploratory data analysis
- model building
- 'what if' calculations.

## Danger — thin ice

As with most areas of human endeavour, power brings with it risks and dangers, and spreadsheets are no exception. A particular problem with spreadsheet construction is that it can be an undisciplined activity. The ease with which formulae can be assembled makes it tempting to put together an application as quickly as possible — usually omitting many of the elements of defensive coding that a professional programmer would include as a matter of course.

This can lead to considerable problems, especially for laboratories operating under quality management systems such as Good Laboratory Practice (GLP) or Good Manufacturing Practice (GMP), for which it is necessary to be able to demonstrate the validity of results obtained using computerized systems. Even for laboratories in which no such formal requirement exists, it is within the users own interests to be able to provide evidence of the correct operation of any computer system involved in the production of analytical results.

As we saw in the introduction, when you see a colleague using a spreadsheet and it matches your needs, why bother to write one yourself? Instead copy the spreadsheet and compound the problem!

The spreadsheet program itself, including the functions, will have been designed and coded by a team of professional programmers who will (or should have!) followed a quality assurance protocol for software development. A program designer would worry about questions such as what happens if an input variable to my function receives the wrong type of data? For example, a function to raise a number to a power would expect to receive two numeric values — the number to be operated upon and an exponent. If a user were to type in text in place of one, or both, of these numbers the function clearly could not work.

Similarly, a function to extract a square root would be expected to work on positive numeric data but if supplied with a negative number would be unable to produce a result. If the programmer were simply to ignore these potential problems his code would be guaranteed to crash when the problems eventually arose (as they surely would) and bring his program to an ignominious halt. To avoid this happening, professional programmers build in defensive code or error-handling routines to test for eventualities of this sort. Such code generally alerts the user to the nature of the problem and provides an opportunity to recover from the error.

We said earlier that people did not need to become programmers in order to create spreadsheet applications. However, this is not entirely true. In fact, anyone who creates a spreadsheet application is effectively writing an extension to the spreadsheet program, customizing the program for his or her particular purpose. So anyone who creates spreadsheet applications is actually engaging in programming whether they realize it or not.

The danger is that the casual developer of a spreadsheet application may not understand general computer programming principles and hence could unwittingly create problems for future users of their application. It's a bit like designing and building a house.

- An architect will produce drawings of what the finished house will look like, working within building regulations and codes of practice.
- A builder will transform the diagrams into the finished article, but will liaise with the architect over interpretation of specific design issues. In addition, the builder will dig the foundations to the correct depth — as specified by the building codes and design specifications. Then the builder will fill them with the appropriate material, again specified in the design plans and building standards.
- Independent inspections by competent authorities will subject the work to checks between the design and practice, as well as ensuring that the appropriate codes of practice have been followed. These inspections will be at critical stages of the building; for example, after digging the foundations, filling the foundations, before the roof is put on and a final inspection before occupation (use) of the structure.

You can see how this equates to quality software development.

In comparison, along comes the casual developer who will be keen to get on with the job and will not always see the necessity of doing things that do not directly impact on the problem at hand. For example, why bother with the architect? Also, you can build a wall without foundations; however, it will not stay up for very long. Such is life with many spreadsheets.

For all the care they take, however, professional programmers still make mistakes and, despite all the testing and inspections that are made, virtually all professionally written software contains anomalies, errors, bugs or 'features'. For the most part these errors are relatively minor or manifest themselves only under unusual circumstances — as the more serious errors should have been found and corrected during the testing stage. If it is so difficult for the professional to write error-free code, it is not hard to see how the casual application developer can unwittingly create problems when let loose in such a seductively simple programming environment as a spreadsheet. Getting worried? You should be and we haven't yet mentioned the contribution an application user can make to the potential mayhem[1]!

## No_Problems@Itcanthappentome.com

As we mentioned in the introduction, these spreadsheet problems always happen in other organizations.

*Example 1:* An organization automated their dissolution calculations using a spreadsheet. After submitting a licensing application, someone thought it would be a good idea to check whether the calculations were correct. They were not. A substantial report had to be written, detailing the extent of the problem and the impact of the error. This was then submitted to the regulatory agency and delayed the issue of the marketing licence.

*Example 2:* Purefac Pharmaceutical Company was inspected by the Food and Drug Administration in October 1994[2] who wrote the following observation in the Form 483: "No written SOPs for validating spreadsheet macros to calculate potency for dissolution, content uniformity and assay. Documentation submitted by the firm to show the validity of two spreadsheet programs that were used to calculate dosage uniformity by weight variation generates inaccurate data. The relative standard deviation calculation has a ±3% error that may be due to an inaccurate formula or inaccurate step in the programming process." The follow up investigation led to the issue of a warning letter on 23 November 1994.

So, what can be done to limit the damage? In essence, our concerns centre on quality assurance. We need to have confidence that the results produced by the application are correct and fit for purpose. Leaving aside the issue of deliberate fraud and/or sabotage which, fortunately are not that common, our main concerns are with ensuring that an application has been correctly 'programmed' and adequately tested, and that a user of the application cannot be misled by it into making mistakes. There are thus three aspects to consider.

- Design: defining the scope and calculations to set up a spreadsheet application.
- Testing: does it work correctly as defined in the design?
- Control: who can use it and who can change it?

## Design: No_URS@Dontdoit.com

Design covers anything that affects the way an application appears to the user, as well as all those things that affect the way it works. So, deciding which cell to put a particular datum in is a design issue as is the way in which that cell is to be formatted. Design factors have a strong influence on the likelihood of obtaining correct results from any computerized procedure.

Incorrect results with spreadsheet applications arise for a number of reasons, the main ones being

- errors in the spreadsheet program's built-in functions
- mistakes by the application developer in setting up or coding the application
- mistakes by the user caused by poor design
- carelessness on the part of the user.

There is not much a developer can do about the last of these. Therefore, this article concentrates on the second and third (although if the developer becomes aware of the first he may be able to devise a workaround, hence the advisability of extensive testing).

Deciding which cell to put a datum in may seem trivial but it can have important consequences.

Spreadsheet applications often extend beyond the visible area of the computer screen and, essentially, exist as virtual surfaces with different regions of a surface being brought into view by means of the scroll buttons. When a spreadsheet application is first loaded, the upper left-hand corner is normally displayed. With poor interface design, it is possible for a new user to be misled into thinking that all of the application is contained within the initial viewable area. This in turn can lead to them omitting to enter required data in cells located outside of the immediately viewable area (it has happened!).

We suggest an overall map of the spreadsheet as a first stage in the design process that would highlight the key stages of the calculations to be performed (Figure 1).

- name, objective, version number and procedure reference
- cell references for entering experiment details
- standards used in calculations, preparation details etc
- time points
- experimental observations
- calculations
- comparison against specifications, limits etc
- overall results
- links with other spreadsheets etc.

The aim is to agree, on a single page, the whole scope of the spreadsheet calculation. In addition, all of the functional needs should be written out, especially calculations, in a URS[3].

A note of caution, however, as organizations become global, spreadsheets don't. For organizations and individuals wishing to develop global spreadsheet applications, the following is a cautionary tale. The UK version of Excel 97 is not identical to Scandinavian versions, as two of us have found running courses on computerized system validation over a number of years using an Excel spreadsheet as an example. Calculations that work well in the UK version do not run correctly in Scandinavian versions of the same program. Therefore, it is highly unlikely that templates and macros will migrate seamlessly across language versions either. One way around this is to agree on a common language version across all sites. This is great in practice but getting agreement first is difficult.

## Titanic@Iceberg.com

With luck the developer should have anticipated any problems regarding poor

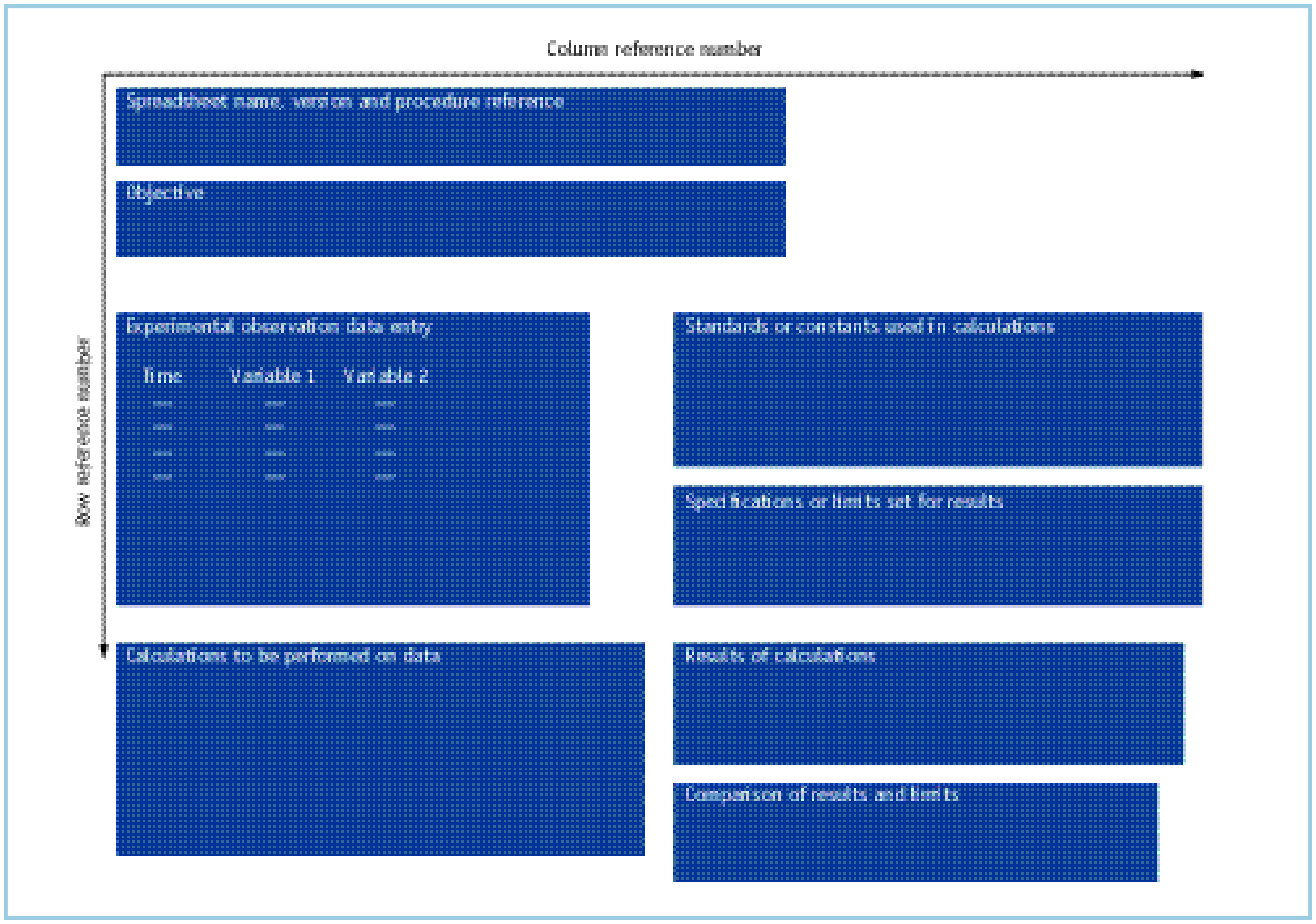
Column reference number
Row reference number
Spreadsheet name, version and procedure reference
Objective
Experimental observation data entry
Time
Variable 1
Variable 2
Standards or constants used in calculations
Specifications or limits set for results
Calculations to be performed on data
Results of calculations
Comparison of results and limits

Figure 1 Outline spreadsheet design.

entry of data, and the spreadsheet application will generate an error message and stop working. However, the effect can be more insidious. As an illustration, if a missing datum was intended to be added to, or subtracted from, the result of an intermediate calculation, then it would simply result in 0 (zero) being added or subtracted. In other words the final result of the calculation would only be a partial result but this would not necessarily be obvious.

It is not possible to list all of the potential problems that could arise as a result of poor interface design and/or coding. Even if such a list were possible, it could never be complete. It is also impossible, in an article of this length, to provide a comprehensive guide to spreadsheet design but, from the example given, it is clear that the casual application developer would benefit from working to a set of documented design principles[4].

## Avoiding_Iceberg@Smartdeveloper.com

To minimize any potential problems that the finished application may cause for the user, it is important to have a clear view at the design stage of what the application is intended to do and, broadly, how it should do it — for example, specify the algorithms it should employ. This calls for a careful analysis of the way the task is currently performed or, if the task is a new one, of how it should be performed. Existing tasks are not always executed in the best way possible (though they may once have been given the resources available at the time) and the design stage of an application provides an opportunity to introduce improvements.

A crucial aspect of design in relation to software is to specify what is required and, equally important, to write this down in the form of a URS. Even if you — the developer — will (at the moment) be the sole user, it is important to record what you intended to do. You never know what data you or someone else might apply to that application in the future. You never know who will walk past your workstation as you are using the system...

While not wishing to suggest that developers of spreadsheet applications should all attempt to become fully fledged computer programmers, we would nevertheless recommend the adoption of the following points of best practice.

- There should be a clear written statement of what the application is intended to do (URS). This should be prepared before starting to build the application. As well as guiding the build, it can be used later to verify that the finished application does what it is intended to do.
- The application itself should be well documented and macros in particular should be liberally commented on. It should be possible for anyone competent in developing applications for the given spreadsheet program to come along, maybe several years later, and rapidly gain a good understanding of what the application is supposed to do and how it works.
- Calculations should be documented when the spreadsheet application is finished so that another programmer or user can maintain it in the future. For example, the way the time points and observations are manipulated will be documented as well as the spreadsheet calculations that result. For business critical spreadsheets, an organization may want to document individual cell calculations and any limits and error trapping routines etc.
- Ideally, and especially for critical systems, the application should check that all input data are complete and of valid type.
- Formulae employed in an application should be documented (with reference to relevant literature sources) and validated as correct and appropriate for the intended use.
- Each application should have a unique version number and should also indicate the version number of the spreadsheet program it was developed on.
- Roles and responsibilities for the user base are defined including the security and access of each.

There are also some particular design issues to think about.

- A spreadsheet should have a clearly displayed title together with an explanation of its purpose.
- To simplify the appearance of an application, sheets that do not accept manual data or present output data (e.g., macro sheets) should be hidden.
- If possible, all manually entered data should be placed within the confines of an initial sheet view. If data are needed elsewhere, they can be copied automatically elsewhere.
- If practicable, it is preferable to use additional sheets for data input rather than requiring the user to scroll large distances to disconnected regions of a single sheet.
- If the screen must be scrolled to complete user input, then this should be clearly indicated on the initial page presented to the user (the users can, for example, be directed to specific cell addresses).
- User input cells can be distinctively colour coded to assist recognition. Preferably a single colour should be used — one should avoid the 'rainbow effect' as this can defeat the object of using colour. Of course, none of your users are colour blind.
- Consideration should be given to providing a separate notes sheet to guide the novice user.

The general philosophy here is to specify the system as fully as possible, thereby making it as difficult as possible for the user to make a mistake. This involves giving careful thought to those areas that might confuse a user and designing accordingly. At the end of the day there is no substitute for correct definition of the requirements in a URS and then testing against them. As wide a range as possible of potential users should be involved in testing and their opinions sought before an application is released for use. It is often the situation that programmers/developers are too close to their work to see the obvious, and it is surprising how many useful improvements can be suggested by people not directly involved with development.

### Testing@Safetynet.com

Having constructed an application it will need to be thoroughly tested before being put into use. One or two data sets will of course have been used during development but post development testing needs to be more stringent. There are several points to think about:

- The application should be tested with data for which correct results are already known.
- Several different test data sets should be employed with data spanning the range of expected values, as well as testing any boundaries and limits for data input.
- Where multiple conditional paths through an algorithm exist, particular attention should be paid to testing the control logic at the transition points.
- The outcomes of all tests should be recorded together with any relevant observations or comments.

Changes to one part of an application's code can lead to unexpected effects on another part of the code. To check for this, the application should be retested with the test data sets whenever any alterations are made to it.

### Control (483_Observation @Itsnotmyfault.guv)

Control of spreadsheet applications is important. Spreadsheet applications have a tendency to grow[1]. They are usually developed, initially, to handle a particular task, and their success in this prompts thoughts of extending them for other processing tasks. If, on the one hand, the application has been well documented, and the original developer undertakes the extension, then it has a good chance of being successful.

If, on the other hand, it is further developed by a different person then the likelihood of problems arising increases. This is particularly so when the original application is poorly or inadequately documented. In either situation as an application grows larger and more complex the possibility for errors increases. In addition to minimizing the risk of errors being introduced into an application as a result of further development, control measures are also needed to prevent accidental or deliberate changes being made to the application 'code'.

As with the question of design, some best practice principles for controlling the use of applications can be stated.

- Spreadsheet applications should be subject to conventional software change control procedures. That is to say, no changes to any working application should be made without permission from an appropriate 'authority'. All requests for change must be routed through this authority together with documentation supporting the case for the requested change(s).
- After a change the application should be carefully tested to verify that the change does indeed do what it is expected to do and that there are no unexpected side-effects.
- Any change, however small, requires that a new version number be associated with the altered application.
- A version history should be maintained in an appropriate area of the application.
- Consideration should be given to protecting the application so that a user can make alterations only to cells specifically intended for user input. Where deemed appropriate, such protection should be reinforced by means of passwords.
- Spreadsheet applications should be set up as templates so that the same initial, data-free, environment is presented to the user on each invocation. One should definitely not load a previous instance of the application and change the relevant data — it can be virtually guaranteed that one day a user will forget to change a piece of the old data.
- A periodic review of critical applications to ensure that it still operates under control of the guidelines or regulations pertaining to it.
- Spreadsheets migrated from one version of the application to another must be done very carefully. Look at the release note to see what changes have been implemented and the extent of those changes. If you have macros, the way of running may have changed in the upgrade, which may impact the way they run.

### Spreadsheets and Year 2000 conformity

When using dates with any computer application, the best practice is to use four-digit dates in all instances. Then you'll need to know how your spreadsheet handles dates and if it is Year 2000 compliant.

How does your spreadsheet use date windowing (the method of interpreting the century for any two-figure year)? There can be differences as shown with the Excel spreadsheet.

- Excel 97 uses a date window of 30. Any two-figure date entered that is equal to or greater than 30 is treated as 19XX. Any figure less than 30 is assumed to be 20XX.
- Excel 95 and Version 4 (Office Version 4.3) use a date window of 20. Any two-figure date entered that is equal to or greater than 20 is treated

as 19XX, any figure less than 20 is assumed to be 20XX.

Thus, if you use two-figure dates and date calculations, mayhem is assured when you migrate from one version of Excel to another as the date window has been moved by 10 years. Only use four-digit dates — you know it makes sense.

However, if you enter dates in a four-digit date format, you can elect to display the date as two-digits and all calculations are correct — providing, and this is critical, the developer uses only four-digit date manipulation. The problems arise when two-digit date manipulations are used, and here you will start to see the impact of the windowing change if you are calculating over the next 30 years or so.

Strangely, all versions of Excel recognize 1900 as a leap year but only for reasons of 'backward compatibility' (however, it is not stated if this is with other applications or the programmers who wrote the original version).

For many uses of a spreadsheet, dates are typically used only as markers for when the experiment or data analysis was done. They can be embedded in the header or footer for a printout or entered into a cell with no further date calculations. Many of these dates are two digits and can be fixed by changing the input cell or field. Where spreadsheets are served centrally, the Information Sciences group can help to ensure Year 2000 compliance by setting the default date field to four digits. Globally there is the old problem of different date formats in the US and Europe. One way to overcome this is to agree the date format to be used globally, as at least one organization has done, or to use US military date format, for example, 31-DEC-1999, for no misunderstandings or arguments.

Macros and templates can provide date problems especially if the calculations or manipulations use only two digits. Here you run into issues of the version of Excel the macro was originally written in and if standard or non-standard functions were used. Here you'll need to test the macro to see what impact the date changes will have. Don't forget to include the leap year testing!

## Interfacing with other spreadsheets and data transfer between other applications

Some spreadsheet applications can be set up to pass data including dates to other spreadsheets if required. Data import and export can also use spreadsheets as a transfer medium or for further calculation. Here you'll extend your investigations of date use to ensure that only four-digit dates are used. The size of the task will depend on the complexity of the interfacing and if and how date calculations are used.

## Summary

The great versatility of spreadsheets explains their enduring popularity, but this very versatility hides a subtle weakness in that it is all too easy for the unwary to build faulty applications. By paying careful attention to the way in which applications are designed and built and by controlling the way in which they are used, it is possible to minimize many of the common problems associated with spreadsheet use. As a final form of safety net, all users of spreadsheets should be encouraged to apply the reality test and ask themselves the simple question... 'do the results look reasonable?'

## Acknowledgement

The preparation of this article was supported, in part, under contract with the Department of Trade and Industry as part of the National Measurement System Valid Analytical Measurement (VAM) Programme[5].

## References

1. R.M. Lindstrom and C. Asvavijnijkulchai, *Fresenius J. Anal. Chem.*, 360, 374, (1998).
2. The Gold Sheet, 30(7), (1996), F-D-C Reports Inc., Washington DC, USA.
3. R.D. McDowall, *Scientific Data Management*, 2(1), 8–19, (1998).
4. R.A. Coad, *Lab. Pract.*, 41(12), 33, (1992).
5. J. Fleming, *Anal. Proc.*, 32, 31, (1995).

*Bob McDowall is Principal of McDowall Consulting, involved in the design and validation of LIMS systems and software applications. He is also Senior Visiting Fellow in the Department of Chemistry, University of Surrey, UK, and a member of the Editorial Advisory Board of* Scientific Data Management.

*Chris Burgess, an internationally recognized expert in standardization and validation of analytical instrumentation, is an analytical scientist with over 20 years' experience in the pharmaceutical industry. Currently he is Principal of Burgess Consultancy.*

*Bill Hardcastle is an analytical chemist working at the Laboratory of the Government Chemist. His interests include assessing the quality of scientific software and attempting to raise the quality of his own code.*